高级类型

学习目标

- 使用 联合类型、交叉类型、字面量类型 来满足更多的标注需求
- 使用 类型别名、类型推导 简化标注操作
- 掌握 类型断言 的使用

联合类型

联合类型也可以称为多选类型，当我们希望标注一个变量为多个类型之一时可以选择联合类型标注，或的关系

```
function css(ele: Element, attr: string, value: string|number) {
    // ...
}

let box = document.querySelector('.box');
// document.querySelector 方法返回值就是一个联合类型
if (box) {
    // ts 会提示有 null 的可能性，加上判断更严谨
    css(box, 'width', '100px');
    css(box, 'opacity', 1);
    css(box, 'opacity', [1,2]);  // 错误
}
```

交叉类型

交叉类型也可以称为合并类型，可以把多种类型合并到一起成为一种新的类型，并且 的关系

对一个对象进行扩展：

```
interface o1 {x: number, y: string};
interface o2 {z: number};

let o: o1 & o2 = Object.assign({}, {x:1,y:'2'}, {z: 100});
```

小技巧

> `TypeScript` 在编译过程中只会转换语法（比如扩展运算符，箭头函数等语法进行转换，对于 `API` 是不会进行转换的（也没必要转换，而是引入一些扩展库进行处理的），如果我们的代码中使用了 `target` 中没有的 `API` ，则需要手动进行引入，默认情况下 `TypeScript` 会根据 `target` 载入核心的类型库
>
> `target` 为 `es5` 时: `["dom", "es5", "scripthost"]`
>
> `target` 为 `es6` 时: `["dom", "es6", "dom.iterable", "scripthost"]`
>
> 如果代码中使用了这些默认载入库以外的代码，则可以通过 `lib` 选项来进行设置
>
> http://www.typescriptlang.org/docs/handbook/compiler-options.html

字面量类型

有的时候，我们希望标注的不是某个类型，而是一个固定值，就可以使用字面量类型，配合联合类型会更有用

```
function setPosition(ele: Element, direction: 'left' | 'top' | 'right' |
'bottom') {
    // ...
}

// ok
box && setDirection(box, 'bottom');
// error
box && setDirection(box, 'hehe');
```

类型别名

有的时候类型标注比较复杂，这个时候我们可以类型标注起一个相对简单的名字

```
type dir = 'left' | 'top' | 'right' | 'bottom';
function setPosition(ele: Element, direction: dir) {
    // ...
}
```

使用类型别名定义函数类型

这里需要注意一下，如果使用 `type` 来定义函数类型，和接口有点不太相同

```
type callback = (a: string) => string;
let fn: callback = function(a) {};

// 或者直接
let fn: (a: string) => string = function(a) {}
```

interface 与 type 的区别

interface

- 只能描述 `object` / `class` / `function` 的类型
- 同名 `interface` 自动合并，利于扩展

type

- 不能重名
- 能描述所有数据

类型推导

每次都显式标注类型会比较麻烦，TypeScript 提供了一种更加方便的特性：类型推导。TypeScript 编译器会根据当前上下文自动的推导出对应的类型标注，这个过程发生在：

- 初始化变量
- 设置函数默认参数值
- 返回函数值

```
// 自动推断 x 为 number
let x = 1;
// 不能将类型“"a"”分配给类型“number”
x = 'a';

// 函数参数类型、函数返回值会根据对应的默认值和返回值进行自动推断
function fn(a = 1) {return a * a}
```

类型断言

有的时候，我们可能标注一个更加精确的类型（缩小类型标注范围），比如：

```
let img = document.querySelector('#img');
```

我们可以看到 img 的类型为 Element，而 Element 类型其实只是元素类型的通用类型，如果我们去访问 src 这个属性是有问题的，我们需要把它的类型标注得更为精确：HTMLImageElement 类型，这个时候，我们就可以使用类型断言，它类似于一种 类型转换：

```
let img = <HTMLImageElement>document.querySelector('#img');
```

或者

```
let img = document.querySelector('#img') as HTMLImageElement;
```

> 注意：断言只是一种预判，并不会数据本身产生实际的作用，即：类似转换，但并非真的转换了